

UNITED STATES PATENT APPLICATION
FOR

METHOD FOR REDUCING AN IMPORTANCE LEVEL OF A CACHE LINE

INVENTOR:

ARIEL BERKOVITS

ପାତ୍ରବିନ୍ଦୁ

2207/6856
EL372085009US

Method for Reducing an Importance Level of a Cache Line

Field of the Invention

The present invention relates generally to a cache associated with a computer central processor unit (CPU), and, in particular, to a method for reducing the importance level of a cache line.

5

Related Technology

As is known, a cache is a fast local storage memory used in computer systems. The cache is typically arranged between the CPU and the main system memory. The cache is used to hold copies of data that are frequently requested from the main system memory by the CPU. A memory system can consist of several levels of caches. The lower the level of a cache, the closer that cache level is to the CPU and the faster and smaller the cache may be.

10 A common measure of cache performance is the "hit rate." When the CPU requests data from the main system memory, the cache control logic checks if the information is available in the cache memory. A cache hit occurs when the information requested by the CPU is in the cache. The cache responds to a hit by passing the requested information back to the CPU. The CPU receives the data relatively fast so it can handle it with a relatively short delay.

15 If the data requested by the CPU is not in the cache, a "miss" occurs. The data requested must then be retrieved from the slower main system memory or from a higher level of cache. A cache may be divided into a number of "lines," or entries. A line of cache may hold data for more than one memory access. Typically, a copy of the retrieved data is saved into the cache memory in a cache line, overwriting the data currently existing in that line. Due to cost considerations, the cache memory is of limited size. Therefore, a 20 so-called replacement policy, or algorithm, is used to determine which line of the cache memory is to be replaced when data is retrieved either from the main system memory or

from a higher level of cache.

The cache-hit rate is defined to be the percentage of memory requests that were completed by accessing the cache without going to higher cache level or to the main memory. High cache-hit rate results in higher overall CPU performance.

5 The replacement policy used by the cache has a direct effect on the hit rate of the cache. For example, replacing data that will be needed subsequently in a given program or process results in a lower hit rate since the CPU will then later not find the needed data in the cache memory. The CPU will have to go to the (slower) main system memory to retrieve the needed data. Thus the replacement policy affects hit rate and, consequently, 10 overall CPU performance.

A variety of replacement policies are known. For example, the least recently used (LRU) policy replaces the cache entry which was less recently used compared to other cache entries. The LRU policy is based on the theory that the least recently the data was used, the less likely the program will request it again. Another replacement policy is the 15 random policy, which selects cache memory locations for replacement at random.

The replacement policy implemented in a given cache is typically fixed in the cache hardware. The application programmer writing software to run on the CPU associated with the cache has no way to provide an indication to the cache that a given line of cache is a good candidate for replacement independent of the particular replacement policy in 20 effect.

Summary of the Invention

The present invention provides a method for reducing an importance level of a line in a memory of a cache, the method comprising providing an instruction to the cache 25 indicating that the line is a candidate for replacement.

Brief Description of the Drawings

Fig. 1 shows a schematic diagram of a cache associated with a main system memory and a CPU according to an embodiment of the present invention;

Fig. 2 shows a table demonstrating prior art cache line replacement for a cache set and
5 memory access sequence for an LRU replacement policy; and

Fig. 3 shows a table demonstrating cache line replacement when a method for reducing an importance level of a cache line according to an embodiment of the present invention is applied to the cache set, memory access sequence, and LRU replacement policy of Fig. 2.

10 **Detailed Description**

Referring to Fig. 1, cache 14 is connected to CPU 12 via bus 18 and to main system memory 16 via bus 20. Instruction storage medium 6 is read by input/output device 8, which feeds instructions stored on input/output device 8 into CPU 12. Instruction storage medium 6 may be any type of suitable medium for storing instructions of a program, such as, for example, a magnetic storage disk. Input/output device 8 may be any type of suitable device such as, for example, a disk drive. CPU 12 may be any type of appropriate CPU, such as a processor or microprocessor. Main system memory 16 may be any type of appropriate storage medium, such as dynamic random access memory (DRAM), for example. Cache 14 includes cache control logic 24 and cache memory 26. Cache 14 may be any type of appropriate cache system. Cache memory 26 may be static random access memory (SRAM), for example. As embodied herein, cache memory 26 is part of a first cache level. Other, higher levels of cache memory may be provided.

An instruction according to an embodiment of the present invention, hereinafter referred to as the reduced importance cache line (RICL) instruction, may be an independent
25 memory access instruction. Alternatively, the RICL may be a part of, or an extension of, another memory access instruction, such as, for example a 'store' instruction. The RICL is decoded by the decoder of CPU 12 and sent to a memory control unit (MCU) associated

with the CPU with an address which is the parameter of the instruction. The MCU then executes the instruction.

As embodied herein, each location in main system memory 16 can map to only a subset of the total number of cache entries, or lines. Each of these subsets is collectively known as a “set.” Control bits associated with a cache set indicate which entry of the set will be allocated for this memory data, replacing a copy of data already in that cache line. As embodied herein, a fixed heuristic function is used as a replacement policy to set the value of the control bits according to the history of memory requests. There is, as is typical, no way to directly control those bits using software.

Reference may now be had to Figs. 2 and 3 to demonstrate how an RICL instruction according to an embodiment of the present invention may be used to decrease the number of memory requests from CPU 12 completed by accessing cache 14 without going to a higher cache level or to main system memory 16, and thereby increase cache hit rate.

Fig. 2 shows a table demonstrating a prior art cache line replacement for a cache set and memory access sequence using an LRU replacement policy. A sequence of eleven memory accesses { a, b, a, c, d, b, b, e, a, c, d} are mapped to the same four-line cache set {0, 1, 2, 3}. Each of { a, b, a, c, d, b, b, e, a, c, d} indicate a main memory location being accessed by the CPU. It is assumed that the cache set {0, 1, 2, 3} initially contains copies of data for locations w, x, y and z, respectively, i.e., cache line 0 corresponds to memory location w, cache line 1 corresponds to memory location x, cache line 2 corresponds to memory location y and cache line 3 corresponds to memory location z.

Columns 30-41 in Fig. 2 represent:

- in row P, the sequence of eleven memory accesses, sequentially from left to right;
- in row Q, the allocation of the memory access retrievals when the memory access required access to the main system memory, i.e., in which cache line of cache set {0, 1, 2, 3} the retrieved data is saved;

- in rows R, S, T, U, the ranking of the cache lines of cache set {0, 1, 2, 3} based on the control bits according to the LRU replacement policy, row R indicating the least recently used cache line, row S indicating the next least recently used cache line and row U indicating the “most” recently used cache line of the set, i.e., least recently used increasing from bottom to top; and
 - in row V, the main memory location for which data was replaced under the least recently used replacement policy.

Initially, cache set {0, 1, 2, 3} contains copies of data for locations w, x, y and z, respectively, and the LRU replacement policy ranking is cache lines 0, 1, 2, 3 (see column 10 30). Upon the first memory access, for main system memory location a (row P, column 31), the data for location w in cache line 0 is replaced with a copy of the data from main system memory location a, since cache line 0 is the least recently used cache line, as indicated by the 0 in row R, column 30. The replacement of data for location w is indicated by the w in row V, column 31. According to the LRU replacement policy, cache line 1 then becomes the least recently used cache line, as indicated by the 1 (column 31) taking the place of 0 in row R. Similarly, upon the second memory access, for main system memory location b (row P, column 32), cache line 1 is replaced with a copy of the data from main system memory location b, since cache line 1 is the least recently used cache line, as indicated, as noted above, by the 1 in row R of column 31. The data for location x is thereby replaced, as indicated in row V, column 32.

Upon the third memory access, for main system memory location a (row P, column 33), the data for location a is already present in cache line 0, so no access of the main system memory, and hence no replacement of a cache line, is necessary.

In the complete access sequence depicted in Fig. 2, it is apparent from row V that a total of eight cache entry replacements are necessary (w, x, z, y, a, b, c, d).

Referring now to Fig. 3, a table similar to that shown in Fig. 2 is presented. Fig. 3 depicts the same memory access sequence, with the same LRU policy, as that shown in Fig. 2. In this case, however, an RICL instruction according to an embodiment of the

present invention is implemented together with the seventh memory access (row P, column 37). The RICL instruction here has the effect of moving the cache line (1) containing a copy of the data for main memory location b to the top of the LRU ranking (row R, column 37). Thus, in the eighth memory access (row P, column 38), the data for b in cache line 1 is replaced (see row V, column 38) instead of the data for a in cache line 0, as with the “pure” LRU replacement policy, as shown in Fig. 2 (see row V, column 38 of Fig. 2).

The RICL instruction might be used as shown in Fig. 3 because the data for main system memory location b will not be used as soon as other data, such as location a, by an application running on the CPU. As a result of location b, rather than location a, data being replaced (see row V, column 38 of Figs. 2 and 3), fewer total cache line replacements, i.e., cache misses, occur. Implementation of the RICL instruction according to an embodiment of the present invention has the advantageous affect in this example of reducing the number of cache entry replacements from eight to five. The result is a higher hit rate and, consequently, improved performance of CPU 12.

An RICL instruction according to an embodiment of the present invention may advantageously be implemented in an application kernel running on CPU 12. For example, CPU performance for a matrix multiplication function could be improved using the RICL instruction. Shown below are two code sequence loops for a matrix multiplication $C = A \times B$, where each line of A is multiplied by all line of B to form the first line of C, then next line of A is multiplied by all lines of B to form the second line of C, etc. Code Sequence I is a basic matrix multiplication loop, while Code Sequence II is the same matrix multiplication loop with use of the RICL instruction.

Code Sequence I

```
25      For (int i = 0; i < SIZE; i++) {
          For (int j = 0; j < SIZE; j++) {
              For (int k = 0; k < SIZE; k++) {
                  // C[i][j] += A[i][k]*B[k][j];
                  Load r1 ← A[i][k];
                  Load r2 ← B[k][j];
```

30

```

        R3 ← r1 * r2;
        Load r4 ← C[i][j];
        R3 ← r3 + r4;
        Store C[I][j] ← r3
5      }
    }
}

```

10

Code Sequence II

```

For (int i = 0; i < SIZE; i++) {
    For (int j = 0; j < SIZE-1; j++) {
        For (int k = 0; k < SIZE; k++) {
            // C[i][j] += A[i][k]*B[k][j];
            Load r1 ← A[i][k];
            Load r2 ← B[k][j];
            R3 ← r1 * r2;
            Load r4 ← C[i][j];
            R3 ← r3 + r4;
            Store.RICL C[I][j] ← r3
        }
    }
    // Assume: j = SIZE - 1
    For (int k = 0; k < SIZE; k++) {
        // C[i][j] += A[i][k]*B[k][j];
        Load.RICL r1 ← A[i][k];
        Load r2 ← B[k][j];
        R3 ← r1 * r2;
        Load r4 ← C[i][j];
        R3 ← r3 + r4;
        Store.RICL C[I][j] ← r3
    }
}

```

35

In Code Sequence II, the RICL instruction, or indication, is asserted for every A line the last time it is used. Lowering the importance of used A and C cells, frees space for more B cells in the cache, decreasing the number of main system memory accesses and thereby increasing the cache hit rate.

40

Thus, an instruction according to the present invention provides information to the

cache about an unneeded cache line. A parameter of the instruction is a memory address. The cache associates the memory address with a cache line if it exists in the cache. The instruction indicates that the memory address will not be used in the near future. Therefore, the importance of the cache line, if any, holding this memory address can be reduced. The information provided by the instruction does not affect the semantics of an application program being run on the CPU associated with the cache, but will provide a useful hint to the cache so as to increase hit rate and, thereby, CPU performance. The instruction will not cause exceptions in the CPU operations.

Execution of the instruction may result in a change in the cache control bits that track memory requests from the CPU so as to optimize the allocation of cache lines. As noted above, a memory access may be smaller than the size of a cache line. The cache control logic may reduce the importance of a cache line based on the first indication to any byte of a cache line, after indication to the entire cache line, or after any number of the bytes in the cache line are indicated to be less important. Alternatively, the cache control logic may ignore an indication provided by the instruction entirely. Additionally, the indication provided by the instruction can propagate to higher levels of cache.

An instruction according to the present invention may be advantageously used in application kernels. As is known, application kernel is a small portion of software that consumes a large number of cycles of the CPU in a typical usage of the application.

Because kernels are typically hand written in assembler language, the developer has the knowledge about the application and the ability to schedule instructions, such as an RICL instruction according to the present invention. An RICL instruction according to the present invention could also be applied in compilers, especially feedback driven compilers, or other interpreter of a higher-level language.

An instruction according to the present invention may reside on any suitable instruction storage medium, such as, for example, a magnetic storage disk, as would be understood by one of skill in the art.

Variations may be made in specific implementations that are within the scope of the

present invention. For example, a method according to the present invention may be an addition of a hint bit to an existing memory access instruction. The bit indicates that this access is the “last” access, for now, to this memory location and the corresponding cache entry is a good candidate for replacement. It should also be emphasized that, although an LRU replacement policy was described herein, a method according to the present invention may be applied with any suitable replacement policy and/or cache allocation methodology. An instruction according to the present invention provides an indication that a cache line is a candidate for replacement. The cache control logic may use the instruction to alter the cache allocation methodology in other ways besides mere replacement of a cache line, as would be understood by those of skill in the art.

10

00100100